Crystal Live SDK User Manual

Version 3.7.9

Contents

Getting Started 3

System Requirement 3

Security Model 4

Developer Guide 4

Step 1. Create a project 4

Step 2. Add a service reference 5

Step 3. Bind CL SDK Interface 6

Step 4. Create a callback from CL SDK Interface 6

Step 5. Login Extension 8

Step 6. Add custom fields when record start/stop 8

Step 7. Play or Download record 8

Sample Code 9

API Reference 12

Getting Started

Welcome to the Crystal Live Software Development Kit (SDK). This SDK contains a wealth of resources which are designed to help you build powerful applications using Crystal Live. It is a guide for developers who write server-side code, custom business logic, custom workflow modules and more.

The Crystal Live SDK includes documentation that covers a wide range of instructive and practical information. Before you can use this SDK effectively, make sure that you have Crystal Live Software installed.

Take advantage of SDK-based development in these ways:

(1) You can build a target for updating recording information according information on your system. This allows you to deliver software that provides new value to recordings.

(2) You can build a target to start and stop recording at any time.

(3) You can build a target for getting, searching and playing recordings.

System Requirement

Supported Operating System:

Windows Server 2008, Windows Server 2003 Service Pack 2, Windows Vista, Windows 7, Windows XP Service Pack 2

Supported Platforms:

x86 and x64

You must install this software prior to using the Crystal Live SDK.

a. Microsoft .NET Framework 3.0+

b. Microsoft Visual Studio 2008+

Security Model

Crystal Live provides you with a security model that protects recording files. You cannot play directly records which were encrypted on the server. But you can use Crystal Live SDK to play or download decrypted records.

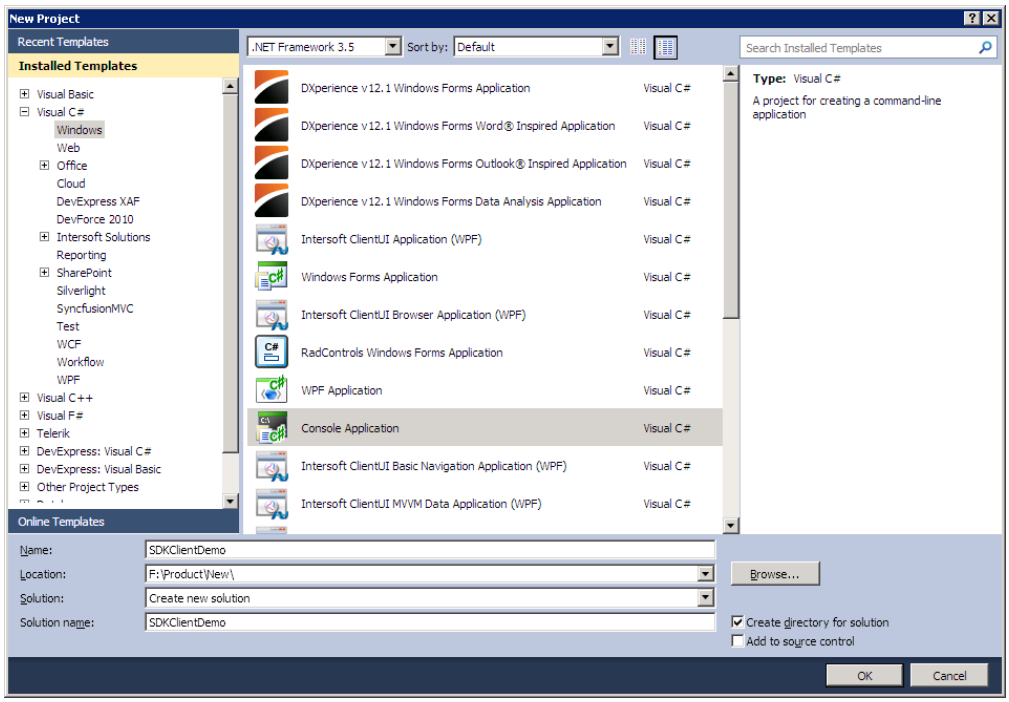
Developer Guide

This chapter describes and explains how to set up your project to use SDK.

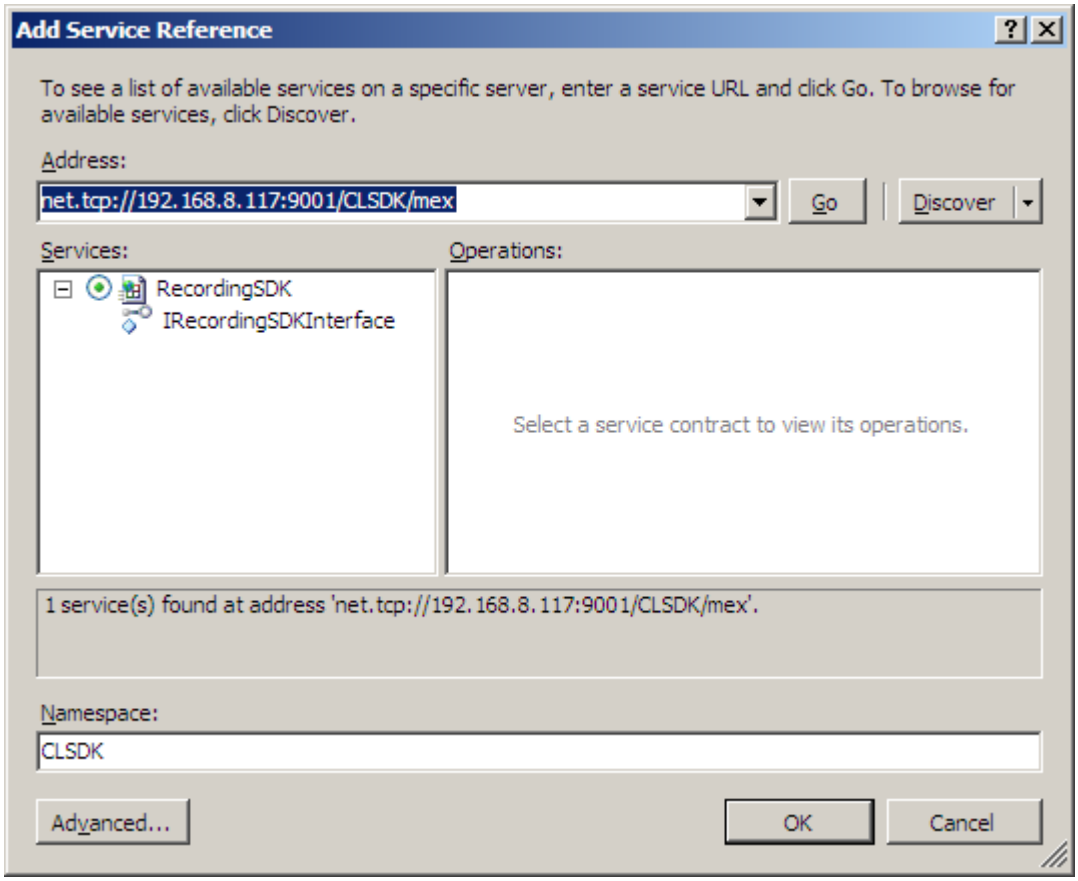
Crystal Live SDK is a Web Service-Based SDK. SDKs are implemented using REST (Representational State Transfer), SOAP (Simple Object Access Protocol) and Net TCP.

To develop a SDK-based client application:

Step 1. Create a project



Step 2. Add a service reference



Please change IP and Port of Address to your CL IP and Port.

Step 3. Bind CL SDK Interface

Please the following picture. You can copy the code to your program and change the ip address and port.

```
public void BindSDKInterface() // Bind CL SDK Interface Link
{
    sdkCallback = new CLSDKCallback();
    BinaryMessageEncodingBindingElement binaryElement = new BinaryMessageEncodingBindingElement();
    binaryElement.ReaderQuotas.MaxStringContentLength = int.MaxValue;
    binaryElement.ReaderQuotas.MaxNameTableCharCount = int.MaxValue;
    binaryElement.ReaderQuotas.MaxDepth = int.MaxValue;
    binaryElement.ReaderQuotas.MaxBytesPerRead = int.MaxValue;
    binaryElement.ReaderQuotas.MaxArrayLength = int.MaxValue;
    binaryElement.MaxWritePoolSize = int.MaxValue;
    binaryElement.MaxSessionSize = int.MaxValue;
    binaryElement.MaxReadPoolSize = int.MaxValue;

    TcpTransportBindingElement tcpElement = new TcpTransportBindingElement();
    tcpElement.MaxBufferSize = int.MaxValue;
    tcpElement.MaxReceivedMessageSize = int.MaxValue;
    CustomBinding binding = new CustomBinding(
        binaryElement,
        tcpElement);

    SDKClient = new RecordingSDKInterfaceClient(new System.ServiceModel.InstanceContext(sdkCallback),
        binding, new EndpointAddress("net.tcp://192.168.8.117:9001/CLSDK")); //CL SDK IP and Port
}
```

Step 4. Create a callback from CL SDK Interface

```
[CallbackBehavior(ConcurrencyMode = ConcurrencyMode.Multiple, UseSynchronizationContext = false)]
public class CLSDKCallback : IRecordingSDKInterfaceCallback // You must implement this callback
{
    /// <summary>
```

Please implement the IRecordingSDKInterfaceCallback, you can see 2 methods (RecordingStarted and RecordingStopped) which are callback from SDK.

```
/// <summary>
/// Recording Started Callback
/// </summary>
/// <param name="RecordID">Every Record has a unique ID</param>
/// <param name="Extension">Extension</param>
/// <param name="RecordingInfo"></param>
public void RecordingStarted(string RecordID, string Extension, RecordInfo RecordingInfo)
{
    Program.MainForm.textResult.AppendText(string.Format("Recording Started RecordID:{0}, Extens
    List<string> customfields = new List<string>();
    customfields.Add("10000"); // Add UserID
    customfields.Add("20000"); // Add WorkerID
    Program.MainForm.SDKClient.AddRecordInformation(RecordID, Extension, string.Empty, string.Em
}
```

```
/// <summary>
/// Recording Stopped Callback
/// </summary>
/// <param name="RecordID">Every Record has a unique ID</param>
/// <param name="Extension">Extension</param>
/// <param name="RecordingInfo"></param>
public void RecordingStopped(string RecordID, string Extension, RecordInfo RecordingInfo)
{
    Program.MainForm.textResult.AppendText(string.Format("Recording Stopped RecordID:{0}, Exter
}
```

This 2 callbacks will output 3 parameters.
RecordID is a unique id for every record, you can use this RecordID to add custom fields.
Extension is an extension number.

RecordingInfo is a call detail information. Please see following description.

RecordInfo Object description

Record Object

Field	Type	Description
Reference	String	Unique ID
Extension	String	Extension
UserName	String	Username
AgentID	String	Agent ID
Direction	String	Call In/Call Out
CallerID	String	Caller ID
CalledID	String	Called ID
StartTime	DATETIME	Start time
StopTime	DATETIME	Stop time
Duration	INT	Duration
FileName	String	Record File name
UploadPath	String	Path of playing or downloading record
ReservedOne - ReservedForty	String	40 Custom fields

Step 5. Login Extension

```
private void button1_Click(object sender, EventArgs e)
{
    BindSDKInterface();
    SDKClient.Login("301"); // Login one extension
    textResult.AppendText(string.Format("Login: {0} \r\n", "301"));
}
```

Step 6. Add custom fields when record start/stop

```
public void RecordingStarted(string RecordID, string Extension, RecordInfo RecordingInfo)
{
    Program.MainForm.textResult.AppendText(string.Format("Recording Started RecordID: {0}, Extension: {1} \r\n", RecordID, Extension));
    List<string> customfields = new List<string>();
    customfields.Add("10000"); // Add UserID
    customfields.Add("20000"); // Add WorkerID
    Program.MainForm.SDKClient.AddRecordInformation(RecordID, Extension, string.Empty, string.Empty, string.Empty, customfields.ToArra
}
```

1. Create a string array of custom fields
2. Call AddRecordInformation of SDK, you must send RecordID which get from RecordingStarted/RecordingStopped and last parameter is custom field which you can send the string array of your fields.

Step 7. Play or Download record

You will get the UploadPath of RecordInfo to generate a http link to play record or download record.
http://ServerIP:4511/RecordInfo.UploadPath;Save=0 to play record file with media player.
http://ServerIP:4511/RecordInfo.UploadPath;Save=1 to download the recording file.

Sample:
http://127.0.0.1:4511/V=1;P=D:\Records\20130624\226\A_SH_SWCIS101253025_20130624095719.wav;CMID=CIS-TEST;Save=0 to play record file with media player.
http://127.0.0.1:4511/V=1;P=D:\Records\20130624\226\A_SH_SWCIS101253025_20130624095719.wav;CMID=CIS-TEST;Save=1 to download the recording file.

Note: No matter the recording file is encrypted or not. When you download or play record, it is not encrypted. Because system will decrypt the record during the download or play.

Sample Code

```
    public partial class Form1 : Form
    {
        public RecordingSDKInterfaceClient SDKClient = null;
        public CLSDKCallback sdkCallback = null;
        public string RecordID = string.Empty;
        public string Extension = string.Empty;
        public Form1()
        {
            InitializeComponent();
            System.Windows.Forms.Control.CheckForIllegalCrossThreadCalls = false;
        }

        public void BindSDKInterface() // Bind CL SDK Interface Link
        {
            sdkCallback = new CLSDKCallback();
            BinaryMessageEncodingBindingElement binaryElement = new
BinaryMessageEncodingBindingElement();
            binaryElement.ReaderQuotas.MaxStringContentLength = int.MaxValue;
            binaryElement.ReaderQuotas.MaxNameTableCharCount = int.MaxValue;
            binaryElement.ReaderQuotas.MaxDepth = int.MaxValue;
            binaryElement.ReaderQuotas.MaxBytesPerRead = int.MaxValue;
            binaryElement.ReaderQuotas.MaxArrayLength = int.MaxValue;
            binaryElement.MaxWritePoolSize = int.MaxValue;
            binaryElement.MaxSessionSize = int.MaxValue;
            binaryElement.MaxReadPoolSize = int.MaxValue;

            TcpTransportBindingElement tcpElement = new TcpTransportBindingElement();
            tcpElement.MaxBufferSize = int.MaxValue;
            tcpElement.MaxReceivedMessageSize = int.MaxValue;
            CustomBinding binding = new CustomBinding(
                binaryElement,
                tcpElement);
            binding.ReceiveTimeout = TimeSpan.MaxValue;

            SDKClient = new RecordingSDKInterfaceClient(new
System.ServiceModel.InstanceContext(sdkCallback),
                binding, new EndpointAddress("net.tcp://" + textBox1.Text +
":9001/CLSDK")); //CL SDK IP and Port
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            if (SDKClient == null)
            {
                BindSDKInterface();
            }
            SDKClient.LoginAll(); // Login all extensions
            Program.MainForm.textResult.AppendText("Login \r\n");
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (SDKClient != null)
            {
                SDKClient.CloseAll();
                Program.MainForm.textResult.AppendText("Logoff \r\n");
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            List<string> customfields = new List<string>();
            customfields.Add("616161");
            customfields.Add("616167");
            Program.MainForm.SDKClient.AddRecordInformation(RecordID, Extension,
string.Empty, string.Empty, string.Empty, customfields.ToArray());
        }

    }

    [CallbackBehavior(ConcurrencyMode = ConcurrencyMode.Multiple,
UseSynchronizationContext = false)]
    public class CLSDKCallback : IRecordingSDKInterfaceCallback // You must implement this
callback
    {
        /// <summary>
        /// Recording Started Callback
        /// </summary>
        /// <param name="RecordID">Every Record has a unique ID</param>
        /// <param name="Extension">Extension</param>
        /// <param name="RecordingInfo">Call details</param>
```

```
        public void RecordingStarted(string RecordID, string Extension, RecordInfo
RecordingInfo)
        {
            Program.MainForm.textResult.AppendText(string.Format("Recording Started
RecordID:{0}, Extension:{1}, UploadPath:{2} \r\n", RecordID, Extension,
RecordingInfo.UploadPath));
            List<string> customfields = new List<string>();
            customfields.Add("616161");
            customfields.Add("616167");
            Program.MainForm.SDKClient.AddRecordInformation(RecordID, Extension,
string.Empty, string.Empty, string.Empty, customfields.ToArray());
        }

        public IAsyncResult BeginRecordingStarted(string RecordID, string Extension,
RecordInfo RecordingInfo, AsyncCallback callback, object asyncState)
        {
            throw new NotImplementedException();
        }

        public void EndRecordingStarted(IAsyncResult result)
        {
            throw new NotImplementedException();
        }

        /// <summary>
        /// Recording Stopped Callback
        /// </summary>
        /// <param name="RecordID">Every Record has a unique ID</param>
        /// <param name="Extension">Extension</param>
        /// <param name="RecordingInfo">>Call details</param>
        public void RecordingStopped(string RecordID, string Extension, RecordInfo
RecordingInfo)
        {
            Program.MainForm.textResult.AppendText(string.Format("Recording Stopped
RecordID:{0}, Extension:{1}, UploadPath:{2} \r\n", RecordID, Extension,
RecordingInfo.UploadPath));

            List<string> customfields = new List<string>();
            customfields.Add("111111");
            customfields.Add("222222");
            Program.MainForm.SDKClient.AddRecordInformation(RecordID, Extension,
string.Empty, string.Empty, string.Empty, customfields.ToArray());
```

```
        }

        public IAsyncResult BeginRecordingStopped(string RecordID, string Extension,
RecordInfo RecordingInfo, AsyncCallback callback, object asyncState)
        {
            throw new NotImplementedException();
        }

        public void EndRecordingStopped(IAsyncResult result)
        {
            throw new NotImplementedException();
        }
    }
```

API Reference

Interface:

Type	Name	Description
void	Login(Extension)	Login one extension to monitor its start and stop
	LoginAll()	Login all extensions to monitor its start and stop
	AddRecordInformation (RecordID, Extension, AgentID, CallerID, CalledID, CustomInfos)	Modify extension, agentid, callerid, calledid or custominfos for one record by recordid.
	Logoff(Extension)	Logoff one extension to stop monitor its start and stop
	CloseAll()	Close all extensions to monitor its start and stop
Callback	RecordingStarted(RecordID, Extension, RecordingInfo)	When recording start, call details will be pushed.
	RecordingStopped(RecordID, Extension, RecordingInfo)	When recording stop, call details will be pushed.

RecordInfo Object/ Record Database

Field	Type	Description
Reference	String	Unique ID
Extension	String	Extension
UserName	String	Username
AgentID	String	Agent ID

Direction	String	Call In/Call Out
CallerID	String	Caller ID
CalledID	String	Called ID
StartTime	DATETIME	Start time
StopTime	DATETIME	Stop time
Duration	INT	Duration
FileName	String	Record File name
UploadPath	String	Path of playing or downloading record
ReservedOne - ReservedForty	String	40 Custom fields